AFRL-IF-RS-TR-2003-133
Final Technical Report
June 2003

UNIFIED MODELING LANGUAGE (UML) FOR INFORMATION ASSURANCE (IA)

Trusted Information Systems

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J758

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** JUNE 2003	**3. REPORT TYPE AND DATES COVERED** Final Apr 00 – December 02

4. TITLE AND SUBTITLE UNIFIED MODELING LANGUAGE (UML) FOR INFORMATION ASSURANCE (IA)	**5. FUNDING NUMBERS** C - F30602-00-C-0081 PE - 63760E PR - IAST TA - 00 WU - 13
6. AUTHOR(S) Brent Whitmore and Brian Appel	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Trusted Information Systems, Network Associates Laboratories, Network Associates, Incorporated, 15204 Omega Drive, Suite #300, Rockville Maryland 20850	**8. PERFORMING ORGANIZATION REPORT NUMBER** N/A
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency, 3701 North Fairfax Drive, Arlington Virginia 22203-1714; AFRL/IFTB, 525 Brooks Road, Rome New York 13441-4505	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER** AFRL-IF-RS-TR-2003-133

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Nancy A. Roberts/IFTB/(315) 330-3566/ Nancy.Roberts@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 Words)***

The IA UML project sought to design and create a simple but powerful methodology and notation for analyzing assurance within the context of a system analysis and design. With such tools, software developers and information assurance analysts express their common problems and solutions in a fashion accessible to both groups. The project extended the common Unified Modeling Language™ notation to express IA concepts. The report documents the history, developments, and conclusions of the project. It recommends practices for software development and suggests possible future work in this area.

14. SUBJECT TERMS Unified Modeling Language, UML, Modeling, Information Assurance, IA, System Analysis, System Design, Assurance Arguments, Domain Model, UML Profile, Object-Oriented Software Engineering, Design Notations, Analysis Notations, Software Methodology, Assurance Methodology, IAUML	**15. NUMBER OF PAGES** 31
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** SAR

TABLE OF CONTENTS

1. Introduction

Every couple of weeks, or more commonly days now, some software vendor, researcher, or analyst discloses some vulnerability in some mainstream software or system. In typical fashion, the producing company or organization scrambles to create a patch to close the problem as soon as possible. Sometimes, with time and analysis a patched system proves to be worse off than those without it because the patch breaks existing functionality or may add other, possibly more easily exploitable, vulnerabilities into the system.

For the most part, there are two main ways to deal with this problem. The first mentioned, and more common of the two, is "constant vigilance." The term constant vigilance is somewhat misleading however because it assumes that the software or system producer acknowledges and attempts to address security vulnerabilities. Commonly, this is not the case with postproduction analysis on a per incident basis. Even then, the focus may not be on security but in eliminating bugs, which may or may not enable security vulnerabilities. Also, some companies either skimp or completely ignore post-analysis, depending on external groups to do their work for them.

The second way to address vulnerability problems is to design the software or system with security in mind. This is less common because a company either lacks security expertise or does not use it properly.

Therefore, this project proposed a way to increase the effectiveness of both types of solutions, with a focus towards the second. Recognizing the problems of the absence of security in the overall development process (design, implement, deploy, reanalyze) and in software designer-security expert communication, the overall ideal for the project was to devise a means to raise security awareness and facilitate dialog. The project team chose to tackle this issue by merging Information Assurance concepts of security into a common design and analysis tool.

This report specifies the overall goals, direction, analysis, and conclusions of the Information Assurance extensions the team created for the Unified Modeling Language (IA UML project). In addition to addressing the successes of the project, this document proposes future venues and extensions to the projects successful, and reusable, work. The following sections formalize the historical progression of the project. The first portion discusses establishment of original goals, overview on the projects eventual direction, and how both apply to the larger picture of UML analysis and design. Additionally, this report documents some of the methodologies developed for the IA extensions and presents an overview of the goals and achievements of the extensions.

Finally, the document includes discussion of potential future work that would enhance changes for success of these techniques and other recommendations of the IA UML team.

2. Original Goals

A current issue with software development and system deployment today is that most security aspects either take a back seat to functionality/convenience, or remain absent due to ignorance/laziness. Arguably most security issues derive from lack of planning and focus during the early design and development stages that may then be compounded by factors like compressed business cycles and "patch it later" mentalities. Another problem during these critical early stages is the possible disconnects between software developers and security experts. Expression and analysis of security properties in the contextually rich language of a software design rarely happens. In today's world, software developers use design notations to describe their system analysis and resulting designs. The Unified Modeling Language™ (UML) is the most commonly used of these notations.

Therefore, the overall goal of this project was to take the widely used, industry standard UML[1] and extend it with an Information Assurance model (i.e. security issues in the context of IA) to remedy security and development communication issues. Our belief was that, by taking advantage of UML extensibility, this project would help define a way to choose, apply, and illustrate IA concepts in UML that could evolve or contribute to a standard notation and methodology. The creation of an IA-UML Domain Model and a corresponding IA-UML Profile would realize this goal, providing IA design information and relationships at various levels of abstraction and perspective. For example, with this common language in place, both software and security personnel could convey and incorporate IA concepts into earlier design stages, averting potential future security issues and problems. The language also helps contributors analyze current designs for weaknesses and the impact of subsequent improvements. More specifically, the project's tools enabled IA engineers to capture IA designs, facilitate in-depth analysis for countermeasure tradeoffs, support adversary countermeasure interaction modeling and provide a method to design and conduct IA experiments. Those software professionals, who are familiar with UML and the software development methodologies that use it, readily comprehend the process and its results in the context of a software design. Additional project goals and focus included countermeasure characterization, countermeasure analysis, modeling of adversary-countermeasure interactions, structured IA UML experimentation, and IA community outreach.

The project established several guidelines regarding the creation and development of the IA UML Profile. This was important because success would require an extensible yet conforming IA extension that would both not disrupt other UML aspects and comply with model interchange tools. The following items indicate some of the guidelines:

- **The IA UML Profile should not change significantly from application-to-application.** A major goal of the project was to provide a generalized and common IA UML Profile for practically any application. The Ultra*Log Program was proposed as a candidate to determine this success.

[1] The Object Management Group (OMG) owns the UML trademark and administers the UML standard.

- **The IA UML Profile should be consistent with other UML profiles.** The project desired to only extend and therefore not conflict or cripple any other profiles for the UML 1.4 standard.
- **Design elements should be well formed and consistent with the UML Meta-model.** The elements chosen for inclusion in the IA UML Profile have properties and functions correlated to IA concerns. These elements present in the IA UML Profile mapped to the Meta-model of the UML 1.4 standard.
- **IA design concepts should provide the proper level of abstraction and detail.** IA design notions and elements should express only information that is helpful to a system designer at a specified design level. Clarification or removal of incorrect, extraneous, and non-helpful aspects was attempted.
- **The IA UML Profile should be consistent with model interchange tools.** The design used XML for the IA UML Profile to allow interchange between the various UML tools currently available.
- **The project should advocate IA specific sub-domains derived from this IA Profile.** In order for acceptance and usage, the project yielded a modular design for the IA UML Profile. Thus, designers using the profile can create new elements for specific design requirements that arise.
- **For a complete Domain model, the design should consider several layers of countermeasures within different system boundaries.** In addition to consistency, IA UML Profile development endeavored to address this modeling requirement since it is very practical that a system/software may depend on other possibly external components (e.g. third party middleware which a designer cannot change).

3. History

Several complete and reusable successes occurred despite the project's reduction in scope. Work began on the project in April of 2000. Initially, the bulk of work focused on modeling the security domain. The project group enumerated classifications and elements for common security countermeasures using UML semantics. Each member took a defined set of the resulting Collaborations (e.g., encryption, authentication, and access control) to develop. Collaborations are collections of UML elements, classes, and interfaces that cooperate to provide certain functions or perform certain system actions. In addition, collaboration may include non-system entities such as UML "actors."[2] In the following months, project members worked on elaborating the parts and systems present in / required to implement these collaborations in system design.

The culmination of this work and discussion lead to subsequent realization of a UML Domain Model specification. This Domain Model successfully laid out IA concepts, issues, and possible examples determined for inclusion in the IA UML extension (i.e.

[2] See Section 4: UML for more UML concepts and information.

Profile). Additionally this model contained the countermeasures and concerns for several types of authentication, access control, encryption, and maintenance. At the end of 2000 and early 2001, work began to finish integration into a final Domain Model and started resolving boundary concerns that arose from communication and control issues between model elements and countermeasures.

During February and March 2001, the project proceeded by enhancing the security metamodel to define specific model elements and attributes. The project then undertook the task of mapping these formal attributes, based on IA concepts, onto the extension mechanisms and semantics present in UML. Completion of the IA UML model elements and attributes, i.e., UML stereotypes, tag definitions, tagged values, constraints, etc., produced a preliminary IA UML Profile around April 2001. Around this time, the project group refined the existing IA UML Profile, while an outsourced group built a project to apply and demonstrate the profile. The IA UML group began development of the Books OnLine (BOL) application demo around mid 2001. After completing the current round of auditing on the IA UML Profile, project members prepared presentations on the "IA UML Profile Analysis Technique." Members presented their work to both the Rational[3][4] Users Group Conference and the Ultra*Log project group.[5]

The next planned step was to incorporate the IA UML Profile into development of the large-scale Ultra*Log project, in order to further confirm the merits of this project. This work would provide yet another valid demonstration and additional IA UML Profile enhancement and analysis. However, program-restructuring issues suspended further progress. The remaining time was used to polish the current IA UML Profile and related documentation. The last major task for the IA UML project entailed presenting the project and its associated successes to a technical meeting of the OMG Security Special Interest Group on January 28, 2002. In the remaining time allotted, participants completed this final report for the project.

4. UML

4.1. *What It Is*

As technology develops and demands the creation of new elaborate systems, the software required to run these systems becomes increasingly complex. Using modeling languages during the design, development, and improvement stages helps developers master

[3] Rational Corporation is a prominent supplier of object-oriented software development tools, methodology, and consulting to the software development industry. Three of the company's principals developed the UML
[4] IA UML Profile Analysis Technique presented July 25, 2001 to the Rational Users Group Conference in Denver, CO.
[5] IA UML Profile Analysis Technique presented July 11, 2001 to the Ultra*Log Principal Investigators Meeting.

software complexity.[6] Modeling languages allow designers to define and structure the various code modules present and required for a given project. In the mid-1990's, several modeling languages/conceptions existed such as OMT (Object Modeling Technique), Booch, and Object Oriented Software Engineering (OOSE). Due in part to possible confusion and conflicts associated with transitioning between these languages, desire for a uniform standard produced an effort to incorporate the various modeling languages. The Uniform Method released in October 1995, was one of the early precursors to the Unified Modeling Language (UML). This product was a result of the effort by Grady Booch (Booch), a founder of Rational Software, and Jim Rumbaugh (OMT), who later joined Booch at Rational.[7] Later in the fall of 1995, Ivar Jacobson and his Objectory company joined Rational and the unification effort with the OOSE method. Currently maintained by the Object Management Group (OMG), many other leading methodologists, software vendors, end-users of the various modeling languages contributed (and still do) to the establishment of the UML standard. A common modeling language is important for software engineers, just as blueprints are important for architects. With UML, an understandable architecture permits a clear definition of components and their interactions. Of course with any powerful tool, successful use requires careful and proper application.

UML benefits development by helping maintenance personnel in identifying and tracking problem areas even if the original designers are no longer available. It provides an effectively absorbed written record of a system's design and its relationships to the problems that the system addresses. Although a project's development cycle may entail the same development team throughout its duration, in most cases some members will come and go. Therefore, effective UML usage can help reduce the overhead of educating new members, and accelerate code familiarity with existing members. UML also aids code reuse. Depending on designer policies/facilities, a company or organization may retain libraries of some or all of the implementations written for a given functional component. Because of its complexity, most software development today occurs in smaller teams that are responsible for components. Software components are collections of code that solve a small, well defined set of functions that developers combine to produce full systems or other components. UML not only aids in component design but can also help in implementation by identifying similar, and previously written, components. Using such components that have already been tested and established, cuts down development time. Standardization also offers the benefit of competition between tool vendors, increasing innovation as well as enabling choice and interoperability.

However, UML use is not a necessary condition for software creation. Other, often similar notations are available. UML focuses on providing a standard language for describing analysis and design concepts; it does not directly address how a given effort uses the UML. The standard only deals with semantics for describing system analysis

[6] Although the above indicates software and software-based systems, UML is not confined to software alone and may model hardware, personnel, and other mechanisms.
[7] Feldman, Boris. UML FAQ. http://www.devx.com/uml/umlfaq.asp, accessed July 2002.

and software design concepts along with mechanisms to extend these features to express additional related concepts. Additionally, UML is not a visual programming language nor intended to be.[8] Because some people sometimes prefer diagrams instead of words to describe software relationships, UML grants a standard way to communicate these issues. Diagrammatic notations communicate some concepts, notably relationships between the entities represented in the diagram, more successfully than other techniques.

4.2. Current Uses

Before discussing some common industry practices with, and usage of, UML, a brief overview UML diagrams follows. The current UML 1.4 standard classifies twelve diagram types into three unique categories. The specification[9] defines diagrams for Static Application Structure (four types – Class, Object, Component, Deployment,) Dynamic Behavior (five types – Use Case, Sequence, Activity, Collaboration, Statechart), and Organizational/Management (three types – Packages, Subsystems, Modules.) Each category contains a distinct set of symbols and elements that visually indicate certain aspects of the model, level, and perspective of a given diagram. For example, in structural diagrams a three compartment box symbolizes a class whereas various arrows represent certain inheritance relationships. Dynamic Diagrams use vertical lines to symbolize objects with horizontal arrows indicating certain actions such as messages and methods. Activity Diagrams follow a general top to bottom progression of control/flow with rounded boxes symbolizing Guards (Conditional [true]) and arrows specifying Decisions (Branches). The above is not a comprehensive look at UML functionality. However, it helps to provide some general knowledge of UML.[10]

Structural diagrams are among the most used and helpful of UML's diagrammatic pantheon. Common and popular programming languages like C++ and Java lend themselves to Object-Oriented Programming (OOP) methodologies. OOP handles a problem by factoring parts of the problem into *objects* implemented usually by one or, possibly, a few *classes*. Out of objects, designers can form *components* that are more comprehensive. At these more abstract levels, the UML standard specifies diagrams to accommodate the various views of a given system. Because UML leads itself to programming methodologies, the methods frequently direct designers to generate software designs using UML's structural diagrams. For example, a structural diagram might show a component used for encryption and decryption. These structural diagrams also allow analysis of a given design aiding in solving dependency conflicts and issues, functionality holes, and tight coupling problems.

Designers use UML *dynamic diagrams* to illustrate messaging and algorithmic processes between objects. Dynamic diagrams might illustrate Internet interactions such as client-server request-reply and authentication messaging, e.g., issue and verification

[8] OMG Unified Modeling Language Specification. http://cgi.omg.org/docs/formal/01-09-67.pdf, July 2002.
[9] Introduction to OMG's Unified Modeling Language. http://www.omg.org/gettingstarted/what_is_uml.htm
[10] Holub, Allen I. A UML Reference Card. http://www.holub.com/class/uml/uml.html, accessed July 2002.

functionality of userIDs, credentials, or certificates. Dynamic diagrams model a system's object creation, method calls, and execution flow for a system.

Activity diagrams demonstrate decision-making logic for algorithms and workflow. They differ somewhat from dynamic diagrams since dynamic diagrams usually handle object interactions, e.g., client ←→ server, whereas activity diagrams handle process interactions, e.g., "Did client authenticate itself to the server? If Yes → communicate; If No → disconnect from the client."[11] UML activity diagrams often demonstrate business logic, e.g., "buy low, sell high" algorithms, and manufacturing procedures, e.g., "what to do when" for auto assembly / repair.

4.3. Extensions

One desire in unifying the various pre-existing modeling languages was to keep the UML standard relatively lightweight and simple. However, the effort also incorporated a built-in extension mechanism to the core specification to allow users to *refine* UML functionality to specific needs without breaking the core specification. UML extensions cannot be arbitrary and must follow guidelines that disallow conflicts or contradictions to the core UML semantics. Specifically, the Extension Mechanism subpackage defines these guidelines regarding the usage of stereotypes, constraints, tag definitions, and tagged values for customizations and extensions to the UML base[8]. Extensions direct how one should use stereotypes to define new UML subclasses from the core UML metaclasses. When creating extensions, stereotypes may also define new and unique metaattributes and semantics. Grouping a coherent set of stereotyped, user-defined elements creates a UML Profile.

Constraints, Tag Definitions, and Tagged Values contribute parts that further refine a UML Profile. Constraints allow refinement of model element semantics. Thus, a Constraint can enforce rules in addition to those of the stereotype of a modeling element. This can allow diversification using broad stereotypes instead of multiple but somewhat similar stereotypes. Depending on the Profile, some UML tools may even help in enforcing constraints while modeling. Tag Definitions specify new properties for model elements either as simple type definitions or also as references to other model elements. The definitions help indicate element information and interactions that a stereotype cannot or should not specify. Tagged Values are property values for a model element's definitions. Tag Definitions work at the metaattribute level whereas Tagged Values are dependent on the specific model element.

[11] Mentioned previously, different diagrams can visualize the same area but in different perspectives of a given system design. A client-server software program, for example, could have three distinct but interrelated views based on diagrams from each of the three major categories. For example, consider a design containing three diagrams. A structural diagram shows the classes involved for the client and server objects. A dynamic diagram illustrates the back and forth messaging between client and server objects that are instances of classes shown in the structural diagram. Finally, an activity diagram demonstrates the system's authentication using the messages illustrated in the dynamic and structural diagrams.

A UML Profile itself is a stereotyped package and can contain additional stereotype definitions that augment the UML metamodel class hierarchy (hence, restrictions to prevent name contentions.) Additionally, the Profile may introduce new symbols for stereotypes and other extended model elements to reduce confusion with pre-existing UML constructs. This is important because a UML Profile should be unique enough such that adding orthogonal UML Profiles from other domains to a design does not result in conflicts or contradictions. Effective UML Profiles limit themselves to distinct domains for manageability. A UML Domain Model should address the issues and basis for the creation of its respective UML Profile beforehand. However, not everything mentioned in the Domain Model needs to be present in the Profile. It is possible that several Profiles could have the same Domain Model and vice versa. Consequently, a UML Profile will always remain an extension to the UML core because of the potential for different but overlapping or redundant extensions.

Alternately, one may extend UML through its Meta Object Facility (MOF). The MOF is a way to add new meta-constructs to extend the UML metamodel. As indicated in the UML specification, this ability is "outside the scope and intent of the UML specification." The reason for this is that the MOF contains no restrictions or assurances on the creation or interoperability of a new metamodel. Therefore, an extension to UML will fall under a "lightweight" (and restricted) UML Profile ***or*** an added and possibly incompatible "heavy-weight" MOF metamodel. This presents the axiom that "every profile definition can be an MOF metamodel, but an MOF metamodel might not be a valid UML profile.

4.4. Goals for IA Extension

The main goal of the project was to apply the knowledge of IA experts to create an Information Assurance model as a useful extension to the core UML specification. The goal for the extension was to concisely portray and educate UML system designers to the concepts and analysis mechanisms used by IA experts. Thus, both groups, enabled by the extension, can improve new and current systems through cooperation and common communication.

In order to accomplish this successfully, the proposed IA UML extension consisted of five main goals and stages of attack. Before jumping to design of the IA Profile (the approximate major end goal,) part of the first phase was to get personnel (re)familiarized with the UML concepts and functionality that later stages would incorporate. While learning about UML and its extension mechanisms, the first goal was to design and specify a Domain Model for the IA extension. This Domain Model serves first as a primer to Information Assurance concepts and terminology, especially to the unfamiliar, e.g., software engineers. The model also provided a record of the topics and issues deemed important for an effective and useful IA UML extension. Therefore, even if an unsuccessful or unacceptable IA UML Profile emerged, the same or another group could salvage the Domain Model work for another attempt later. An added benefit to a Domain Model is that it limits the scope of IA problems to a reasonable and agreeable realm while providing technical project documentation at the same time. After completion of preliminary work and the Domain Model, the next step was to design the IA UML Profile itself. Integrating the general IA concepts, laid out in the Domain Model, e.g. countermeasures, into UML model elements completes the overall IA UML Profile. At this second stage, several passes over the preliminary IA UML Profile to add missing features or improve (i.e. simplify) existing model elements occurred. With a mostly-complete Profile, the third goal for the extension was to actually create and demo a sample system in UML using the Profile. The primary reason for this stage was to validate and verify the successes of the preliminary profile. However, in the course of applying the profile, this stage also seeks to develop methodologies in how to effectively use the profile as well as discover any desirable but absent functionality. The final two goals were to (1) further refine the IA UML Profile extension and application processes to a real, large-scale system, and (2) present the project results to appropriate organizations. The team worked on both goals at the same time. So, while perfecting and finalizing the Profile, the project group could portray the project successes and lessen delays for possible adoption and recognition of the IA extensions for UML.

5. Technical Approach

5.1. Methodologies

The first task of the project group was to define an overall methodology (i.e. common way of thinking and plan of attack) for both the Domain Model and Profile constructs (referred to as the IA extensions). Thus, members required a common scope of problems to address and solve. They also needed a way to develop and assimilate their individual work into the work of the group, and to assess that work's success and viability as an extension of Information Assurance (IA) concepts to the Unified Modeling Language (UML). The four main classifications of Vulnerabilities, Threats, Countermeasures, and Assurance emerged from several early group meetings. The IA UML team sought to address each area during the progression of the project and development into an IA UML extension.

As defined by the project team, Vulnerabilities compose the inherent, known or unknown, weaknesses, exploits, breaks, etc. present in a given system or design. In practice, no system is ever free of all vulnerabilities. However, through use of Countermeasures and Assurance Arguments, an adept IA UML user can reduce the overall set of vulnerabilities present both system design and deployment stages. Additionally, model elements that address vulnerabilities, i.e., Countermeasures and Assurance Arguments, may also only seek or succeed in mitigating a given vulnerability. IA mechanisms can reduce the lethality of the vulnerability, e.g., authenticate an unknown user but deny all access by it. IA mechanisms can also render the vulnerability unfeasible, e.g., expensive costs require enormous amounts of time and/or money to take advantage of the vulnerability.

Threats take advantage of Vulnerabilities in the system. Threats can be inside or outside the designed system. For example, a possible external threat might be bombing the facility or physical location of the given system. It may also be a malfunctioning host. Depending on the design, developers might address these types of threats with countermeasures and Assurance that the countermeasure will mitigate certain attacks from a threat.

The primary means of combating vulnerabilities is by using Countermeasures. A Countermeasure, as defined by the IA UML group[12], is a device or mechanism that protects a given system against Vulnerabilities and Threats. In a UML-designed system, the same Countermeasures may operate on various levels and so, be described by different types of diagrams. Also, a system countermeasure does not have to take an all or nothing approach. In most cases, in order to become effective, several model element countermeasures must collaborate to provide the intended protection over the system.

Associated with countermeasures are Assurance entities. In the project's methodology, Assurance can be model elements or assumptions, restrictions or deferrals on either elements or the model itself and arguments – empirical and analytic – for a countermeasure's effectiveness. Specified as a model element, an assurance can collaborate with other model aspects and help shape policy concerns. The latter more general type of Assurance can help reduce the scope of Vulnerabilities to practical and manageable levels. For example, system designers could tag a cryptographic countermeasure with an argument that an attacker does not have access to sophisticated hacking technologies.

As the project team elaborated on the IA extensions, identifying and elaborating typical Countermeasures became a large aspect of its work. Part of the reason for this is that focusing on developing Countermeasures maximizes the primary facet of the project – system security and defense. Additionally, developing pertinent countermeasure model elements requires the corresponding progression of the vulnerabilities that they address. The IA UML team did focus on defining Assurance and Threat model elements until time pressures demanded that they continue on to subsequent phases. In addition, Assurance elements are more of an interim or final step, as opposed to an initial step, of model creation. Although Threat modeling has some important benefits, the project group placed less focus on it because in general it is hard to determine, as well as explicitly define and rank, the methods that an attacker might use to threaten. Threat models help protect against only known attacks, not the unforeseen ones. Instead, the project team logically sought to focus on developing IA extensions for Countermeasures and their covered Vulnerabilities, adding only such Assurance as seemed reasonable in a first "bootstrap" phase. The team felt that they would better address issues like threat and claim analysis and by expanding on assurance as the methodology evolved.

[12] See *Information Assurance UML (IA UML): A Profile for Expressing Security Issues in UML Models*, NAI Labs Report #01-027D.

Building from these categories and assessment of IA conceptions, the project underwent the following three main components:

- IA Collaborations – the generic designs or patterns present in the IA Domain Model used to produce security countermeasures and determine their effectiveness. Address how to use the selected security elements in a design to counteract the expected system threats.
- IA Domain Model – enumerates commonly seen security and system entities as well as their interactions and relationships. Addresses what security elements to consider when designing a particular system.
- IA Profile – describes the elements extended to the UML core model to express the security features of a system as well as display understanding of security issues and concerns. Addresses how to use UML to describe a design's security features and effectively communicate them to other colleagues.

5.1.1. Profile

Development and successful completion of the IA UML Profile by the project team was the concrete means of providing an IA tool for system designers. This profile, denoted "IA-UML," defines how to express the IA information which, whether directly indicated or not, touches all parts of any software and system. Additionally, the Profile conforms to the extension mechanisms of the UML core. Designers can switch IA mechanisms in and out of their designs and analysis at will. Although, in reality, "true" IA is a running system (hardware inclusive) property, augmenting pre-defined IA model elements and models into a system can increase security and information assurance. Building a system from the ground up is not always possible, but designers effectively incorporating the IA UML Profile as early as possible can reap security understanding as attributed benefits. The Profile and its accompanying document[12] provide a standardized way to collect and organize the UML model elements that support IA structure and behavior. It is important to note that the Profile itself does not directly demonstrate secure designs or methods.

The IA UML team's purpose for the Profile was to allow designers to express system designs that implement IA countermeasures. The Domain Model contains the education and application guidelines for model element collaborations. A large part of the project involved defining IA concepts in the Domain Model and then designing appropriate model elements to allow designers to incorporate those aspects into their system designs. Because of the reduced project focus and the Domain Model and Profile development cycle, some IA concepts do not have Profile corollaries. It was also important for the project to validate the IA UML Profile, which consumed additional time and resources. Therefore, some future work on IA enhancements is still available.

The remaining portions of this subsection discuss the factors and functionality present in the IA UML Profile.

The IA UML Profile identifies two tiers of elements, the Core elements and Supplementary elements. The heart of the profile consists of the core, stereotyped elements whereas the supplementary elements are optional (yet recommended.) Any system design done with the Profile should fully utilize all of the core elements. The current IA UML Profile lists the following model elements:

Core	**(Base)**	**Supplementary**	**(Base)**
InfoCon	Class	Authentication Context	Class
Written Policy	Class	Pedigree	Class
Implementation Policy	Class	Sensitive	Association
Policy Domain	Package	Label	Attribute
Countermeasure	Collaboration		
Assurance Argument	Class		

The first core element is the «InfoCon» element, which identifies other classifier elements in the model. This element must contain a state machine responsible for describing system integrity, the information condition of the given system. Typically, states indicate security conditions and offer a range of "normal to threatened to compromised". Proper design of these states is essential so those system countermeasures (object collaborations of the system) can react to changes in information condition[13]. These "attack" levels or "information conditions" help to define eventual security policies to govern individual countermeasure execution. The Profile defines model elements for Policy according to Domain Model concepts.

Policies supply two main functions: rules to govern countermeasures behavior and provide mechanisms to more effectively manage the system (or specific parts of it.) Displayed above, the IA UML Profile contains two types of Policy model elements: Written Policy and Implementation Policy. Written Policy denotes the human readable document that lays out the security requirements as well as the appropriate and required countermeasures for a given system. Implementing Policy is a code manifestation of a Written Policy. An Implementing Policy can identify component collaborations or configuration scripts that drive software system behavior.

However, any individual or group of systems may incorporate several policies at several different levels. Therefore, a Policy will usually specify a Domain, referred to as a Policy Domain, mapped to a UML Package. Because several policies may influence a given model element, an effective design will not always restrict all elements to a certain policy domain package. Thus, the project team decided to only require that «Policy Domain» packages include collaboration definitions. This is because collaborations can reference other participants without constraining them to certain portions in the hierarchy of a model. One piece of the model can own the design while another references it in a separate policy domain. The «countermeasure» stereotype fits into this system by

[13] An example of this could be an auditing countermeasure that increases audit record details during an attack on the system. Detail then decreases when the system identifies the type of attack, e.g., a resource-constrained attack like Denial of Service.

expressing IA behavior through the collaborations indicated in a policy domain. The following list briefly describes the model elements available to a Profile user starting with the Core stereotypes.

InfoCon «infocon» –applies to a class in a given model or level of a system. It contains exactly one state machine describing relevant information condition levels and their transitions. For the stereotyped object, a Statechart Diagram signifies all defined condition levels, which can change other objects in the system and vice versa. Incorporation with UML Interaction Diagrams and Activity Diagrams to detail system behavior while under attack is also an option.

Written Policy «written policy» – depicts a written policy document that can exist as a design or runtime system Constraint, requiring any «policy domain» package to include at least one stereotyped «written policy» class. This class will contain a reference to the actual document as a Uniform Resource Identifier (URI – e.g., http://, file://, etc.) Additionally, when using «implementing policy» stereotypes from the Profile, «implementing policy» classes must be associated linked via an «assurance argument» class to a «written policy» class.

Implementing Policy «implementing policy» – optionally applies to any model element principally involved in countermeasure collaborations. The Profile also identifies references through policy or configuration mechanisms.

Policy Domain «policy domain» – depicts all the relevant elements needed for enforcement of a policy, linked to at least one written policy, and typically existing as a «countermeasure» collaboration or, possibly, an actual class. The latter situation implies inclusion based on sole support of the policies linked to that «policy domain».

Countermeasure «countermeasure» – optionally depicts collaboration behaviors implemented and expressed by one or more countermeasures, each owned by at least one «policy domain» package that in turn references a «written policy» indicating the enforcement methods.

Assurance Argument «assurance argument» – different from class-based elements, these include URIs to documents depicting the evidence surrounding a specific assurance statement on how a defined «implementing policy» adheres to defined «written policy». The Profile defines three types of policy:

- Empirical – usually a set of test cases, plans, or results from a defined operating environment,
- Analytical – usually a mathematical and limited assertion proof demonstrating the specified argument and
- Operating Environment – document(s) that define what environments are applicable.

The following comprise the Profile’s Supplementary stereotypes:

Authentication Context «authentication context» – applies to other countermeasure and design interactions in the model that provide and indicate authentication status for a given

Subject. Although the Profile does not define Tag properties for this element, it does note two Constraints: encrypted, indicating encrypted Subject credentials possibly with a Boolean or (more preferably) the type of encryption; and subject, indicating the authenticated element or entity.

Pedigree «pedigree» – applies to any class with at least one attribute that references some data or resource of another class that the designed system uses to determine integrity characteristics. These classes may or may not rely on other mechanisms to determine software or data integrity. Indicated in the Profile document, trusted sources handle management of pedigrees and their meta-information about model objects and entities.

Sensitive «sensitive» – applies to any model element deemed specially significant and responsible for the overall information assurance of a system. This element is primarily for analysis and examination of system designs to tag important but restricted knowledge about the model such as specific countermeasure collaborations or policy aspects.

Label «label» – applies to an attribute indicating a relationship or implementation of Mandatory Access Control (MAC) countermeasures and collaborations, which utilize the meta-information (i.e., state) of the tagged class for MAC decisions.

Just like the core UML specification, further work can extend Profile mechanisms using concepts from the Domain Model. However, since the Profile currently employs an open framework to apply IA concepts to UML designs, future extension of the Profile requires proper care so that elements remain useful and non-restrictive to UML system designers. The next section further discusses some of the IA topics IA UML Profile model elements can tackle as well as other areas of interest for possible future development.

5.1.2. Domain Model

Although the IA UML Profile was the main result for the project, the team created the Domain Model to act as a foundation for the IA UML Profile.[14] For the duration of the project, the Domain Model developed along the progression of:

- defining the relevant IA concepts and types for inclusion,
- defining those IA Collaborations and interactions, and
- defining the core aspects and principles of the Domain Model that govern the IA UML Profile.

Aside from other mentioned benefits, the Domain Model grants an abstraction from the UML-specific mechanisms which bind a resulting Profile. Therefore, software and system designers intent on including security aspects need not only attempt to apply the IA UML Profile. The Domain Model serves as a complementary IA extension and thus allows UML users the opportunity to develop their own (desirably complementary) IA Profiles depending on their project limitations. Future work and additional IA concept

[14] See *IA UML Domain Model*, NAI Labs Report #01-026D.

incorporation into the Domain Model can strengthen and increase the benefits of this IA extension.

IA UML Domain Model, developed with the Rational Rose UML tool, resides in the Use Case and Logical View in the Rational Rose tool. Therefore, the Domain Model applies to several relevant diagram types available in UML. Diagram types and purpose for each follows:

- Class – illustrates logical structures of a given countermeasure (or set of them).
- Object – illustrates system objects and their relationships.
- Component – illustrates physical structure of the system software under design. [15]
- Deployment – illustrates mapping of the software system to the hardware configuration. [15]
- Use-case – illustrates outside interactions with the system under design. The interactions are not only with users but also with outside systems and other environmental factors.
- Activity – illustrates event flow within a given countermeasure or set of them.
- State – illustrates state and transition behavior. These are usually attached to a class. [16]
- Interaction – illustrates countermeasure behavior with use collaboration and sequence diagrams.

Early design stages for the IA extensions focused on a broad range of initial possibilities for IA related model elements regardless of proposed (but untested) UML complications, e.g., problems posed by policy elements. Through the course of a few meetings, formal concepts and direction for the IA extensions formed.

The project team initially discussed Vulnerabilities and associated Threats, Countermeasures and Assurance Arguments. However, the primary focus of the Domain Model became composing a common set of IA Countermeasures. The model defines a Countermeasure as a "device or mechanism that provides protection to a system against some type of threat." In order to reduce bloat, i.e., redundant, similar, and unwarranted concepts, these were refined to combine topics into common and robust IA elements and countermeasures. Despite a few non-fully functional countermeasures in the Domain Model, the project work addresses these areas, offering insight and a start for possible future work.[17]

[15] Currently the focus level of the Domain Model and the Profile is only on the Logical View.
[16] The InfoCon class is an example of this in the Domain Model.
[17] Because of coupling issues, some of the non-functional concepts presented in the Domain Model have non-functional or absent analogs in the IA UML Profile.

After establishing the IA countermeasures, the next progression was to specify collaborations. Collaborations are the interactions that occur in patterns of typical objects to protect the system from certain threats (that the countermeasures seek to prevent). Therefore, project members analyzed each countermeasure in terms of what other model elements interact with it. Some countermeasures offer different types of collaborations, indicated as subcollaborations in the Domain Model specification. The Domain Model and collaborations specifically address the major IA concepts of Access Control, Authentication, Audit, System Assurance Maintenance, Encryption, and Intrusion Detection. The model also defines the elements of Integrity, Countermeasure, Policy, and Policy Boundary (or Domain) for the IA extension. The following areas indicate these Domain Model IA concerns, as well as their respective collaborations, in the above order.

Access Control

In general, the term Access Control refers to the system that mediates resource access depending on the subject or process. In the Domain Model, Access Control countermeasures protect system integrity and confidentiality by restricting system use and information retrieval. These countermeasures are closely associated with authentication services, which verify and validate the identity of the subject. In the absence or compromise of authentication collaborations, Access Control countermeasures lose their ability to properly mediate access (the subject remains unknown and thus restricted).[18] In the Domain Model two types, or subcollaborations, are represented – Discretionary and Mandatory Access Control (DAC and MAC respectively.) In addition, the Domain Model denotes the Mediator, Access Control Policy, and Authenticated Subject model elements.

Discretionary Access Control (DAC) arbitrates depending on the (usually previously authenticated) identity and or properties of the subject. This subcollaboration type permits the data owners to decide the access policy, which in turn dictates what / how other subjects access that same data. These access policies usually take the form of one of three types that use different ways of classifying the subject, data, and the interactions between them. The first DAC collaboration type is User-Based Access Control. This type grants or denies resource access depending on the identity a subject possesses. In almost all cases, User-Based Access Control uses authentication countermeasures. User-Based decisions are commonly used in access control systems today, e.g., operating systems, where subjects log into a system via a terminal and user account, i.e., the “effective identity.”

The second and usually dominating type of DAC in the Domain Model is Role-Based Access Control (RBAC). In Role-Based access control, individual subjects share common sets of resource permissions with other users. These collections form a Role. Resource owners or administrators assign these Roles to subjects. In the general sense, a

[18] This primarily assumes a consistent access control policy that, either by default or in the absence of an authenticated subject, access is severely limited or outright denied.

subject gains a Role that allows those and only those functions required to perform specified tasks. In a given system, a subject may have multiple roles or even none, relying solely on User-Based mechanisms.

The third type of DAC addressed by the model is Team-Based Access Control. During completion of some project, team leaders create and assign Roles for the project. Subjects then utilize the Roles to access team resources in order to perform their work. Team access control policy may include subjects from many collaborating organizations. So, this form of policy may result in limited and changing access based on timing issues, milestones, redirections, etc. Under team-based access control policy, access is available only when a user is part of a team, and only when the participated role allows.

Mandatory Access Control (MAC) has better controls on delegation and propagation of permissions than DAC. Yet MAC typically coexists with discretionary policies. In this case, a system denies access when either MAC or DAC mechanisms checks deny access. For MAC resources, owners do not determine access rules. Instead, the system imposes a consistent and unavoidable (hence mandatory) access policy for all resources. The Domain Model specifies MAC collaborations as label-based access control, performing mediation depending on those labels attached to subjects and resources. The US Department of Defense orders labels in a hierarchy whereby the subject label must dominate that of the resource. Addition of category or compartment labels to hierarchies can be used to further provide access restrictions.[19] For MAC countermeasures, any detectable attempt to circumvent access control rules prevents access. A key assumption of MAC countermeasures in the Domain Model is that only the security administrator role, and that role alone, controls the group, role, and label membership, assignment, and revocation.

Authentication

In general, the term Authentication refers to the system that verifies and validates the identity of a given object or process. These countermeasures function by observing the properties of the subject. What they are (biotech), what they know (password), and what they possess (identity card) are all viable options in determining identity. In the Domain Model, Authentication ensures truthfulness of identity claims (Fundamental Authentication) and messages transfer (Communication Authentication). However, the model handles each with different collaborations. Authentication also supports additional countermeasures such as Accountability.[20]

The Authentication collaborations found in the Domain Model can apply to both human and computer services as in peer entity authentication. High (strong) assurance

[19] For example, in the general hierarchy of Top Secret → Secret → Confidential → Sensitive, Secret access (i.e. read) usually implies access to resources labeled Confidential as well. In addition, access may require a 'blackhawk' category label from the subject. Commercial labels might include Proprietary, Company Sensitive, or Internal Use Only.

[20] Accountability, another IA concept, deals the functionality that allows enforcement personnel to trace actions to a unique authorizing principal (i.e. determine the authorizing agent of the subject in question).

authentication mechanisms can associate with other Domain Model collaborations like Encryption, e.g., SSL and password encryption.

Most authentication countermeasures are ungated whereby the ability to authenticate is always available and unrestricted. The other form – gated authentication processes – place restrictions on the access and ability to authenticate. For example, with ungated authentication, other countermeasures can undertake burdens like accessibility, such as an unsupervised entry system in an already secured building. In an unsecured building, a gated system might isolate access by only authenticating a single individual at any given time (e.g. secured isolation zone) in order to prevent observation of other subjects. The Access Control collaborations noted above interact with Authentication and its Authenticated Subject model element.

The Domain Model describes four major authentication collaboration types each with some possible minor types. In addition to the Fundamental and Communication collaborations noted above, the model defines Challenge/Response, Sensor, Message, and Session collaborations. Fundamental authentication is the basic form for other, more specific authentication collaborations. Aside from Communication, the authentication collaborations indicated in the Domain Model have added specific techniques and constraints to the base Fundamental collaboration. Countermeasures used for identification can have merit despite lacking a sufficient 'secure' aspect, e.g., userID but no password. Thus, some of the collaborations present in the Domain Model may employ these types of alternative countermeasures.

In the Challenge/Response collaboration, an Authenticator communicates a challenge message to the Subject. Based on some preexisting and known procedure, the Subject then responds according to the challenge message. This process may continue until the Authenticator acknowledges the authenticity of the Subject.[21] An instance of this in the Domain Model, is Password Authentication whereby a system prompts the Subject with an identity claimed and corresponding password.

For Sensor authentication, the Authenticator observes features inherent to the Subject that may or may not involve Subject cooperation or awareness. Systems implementing Sensor type collaborations usually perform both tasks of establishing identity and checking authentication due to difficulty in counterfeiting relevant features.

In the Sensor authentication space, the model lists both Biometric and Token authentication instances. Biometric instances primarily focus on observing the physical characteristics of the Subject in question and matching them against a previous and known record. Token instances observe the physical possessions that the Subject possesses.

[21] Certain implementations might be vulnerable to Replay attacks. This threat occurs when an attacker, despite not knowing a password or other secret, eavesdrops on a successful authentication process and then gains an unauthorized identity by duplicating and retransmitting the process's messages.

Whereas Fundamental authentication deals with direct Subject observations, Communication authentication deals with proving that the Subject authorized some communication actions on his behalf. The two instances of Message and Session authentication for this collaboration are detailed in the Domain Model.

In Message Authentication, an Authenticated Message Sender sends messages to an Authenticated Message Receiver. Authentication relies upon prior transmittal of cryptographic keys to the Key Store of the Receiver from either the Sender Key Store or a trusted intermediary. Relying on cryptographic techniques (e.g. encryption, digital signatures) and restricted key possessions, both sender and receiver have means to verify the message as well as its integrity. From this ability to verify the message, the respective parties are able to authenticate each other.

In Session Authentication, a session joins two Authenticated Connections between Message Sender and Message Receiver. Authentication Clients query their Authenticated Connections to authenticate the messages received through that connection. Similar to Message authentication, Session authentication relies upon prior transmittal of cryptographic keys and associated techniques. Each participant consults their respective Key Stores for proof of identity and exchanges their proofs when they establish a session. Again, cryptographic techniques ensure the proofs are authentic and not tampered with.

Audit

In general, the term Audit refers to a function where the system monitors and collects all the relevant definable events that document the operation and use of that system. The presence and strength of these records collectively determine the Accountability properties of the system. Typically, a system automates the collection of audit events that generated audit log entries in an audit trail, which chronologically details all the security related system activities (and might include other aspects as well). These types of countermeasures are important because they can identify and restrict the misuse or attacks on the given system.

In order to manage collection, Audit policies (possibly using the Policy Model of the Domain Model) govern what, where, and how events are gathered. These events may originate in several systems components such as the operating system, database, or within other countermeasures (e.g. fingerprint reader, firewall). Additionally, the Audit policy can direct how to respond to a single or set of given events (is it security related, should it raise an alarm, what if auditing is down, etc.). The Domain Model document also specifies several model elements such as AuditPolicy, SecurityEvent, AuditLog, and collaborations of System Audit Countermeasures.

System Assurance Maintenance

In general, the term System Assurance Maintenance refers to the system that maintains and coordinates all the countermeasures within a system. Since any system is most likely susceptible to at least some vulnerabilities, the system administration personnel must constantly identify and apply security patches to improve or add countermeasures.

Compounding the problem however, vulnerabilities are typically of a transitive nature. Thus, despite patching countermeasures at one point in time, the vulnerability could still appear and reappear in another countermeasure as well as the patch possibly injecting a new and possibly unknown and undetectable vulnerability into the system. Any efficient design must then recognize this as a fundamental design problem.

In conjunction with operational security, the Domain Model addresses Assurance maintenance in the various mechanisms for policy enforcement and modification. Vulnerability discoveries and changes in the mission of the system may compel updates in policy. In addition, this concept can incorporate "red team" activities such as support and verification as well as increase education and awareness of countermeasures operations.

As indicated in the Domain Model, the Information Condition (InfoCon) model element is an integral part of System Assurance Maintenance. The inclusion of the InfoCon element in a system design affords a more adaptive detection and response when under attack. The Domain Model utilizes InfoCon states in order to enact one or a series of countermeasure policy changes to reduce and terminate a threat. Thus, integration of these collaborations into a design can serve as a foundation to gauge and configure the overall security posture of the system.[22] In the model presented in the Domain Model document, these collaborations require external interactions by vigilant support staff and vendors to augment countermeasure maintenance. In addition, ongoing security education of user and "red team" assessments of the current system and countermeasures, and policies to conduct them, can all help increase system security and readiness.

In the general maintenance hierarchy of the Domain Model, staff falls under the direction of the Command Authority. The Command Authority sets policy and can alter security posture by mandating changes in InfoCon status. These InfoCon elements define state machines for the permitted security postures of the system. In addition, state machines also indicate the permitted transitions from one InfoCon status to another. Therefore, the current InfoCon state is a representation of the current overall policy of a given system. When a state change occurs, all countermeasures enforce the (possibly different) countermeasure configurations in their new state. Proper system assurance maintenance requires us to analyze all possible system states and all the possible transitions among those states. As laid out in the IA extensions, collaborations for Security Assurance Maintenance are complex but very beneficial.

Encryption

In general, the term Encryption refers to the system that protects system integrity, data, and confidentiality. The Domain Model addresses this collaboration through cryptographic countermeasures whose functionality can include one-way hashing (MD4 / MD5) as well as secret, or private, key (DES, IDEA) and public key (PKI solutions)

[22] See the diagram in the Domain Model Document for a more detailed look at System Assurance Maintenance collaborations and applicable model elements as well as what functionality and advantages their inclusion offers. [14]

cryptography. An essential requirement for countermeasures such as these is in proper key management. Key management usually deals with the issue, storage, retrieval, and revocation of cryptographic keys. Depending on the system, the Subject may handle other concerns like key generation and distribution. In most cryptographic countermeasures the security strength depends on the confidentially of some secret, usually a cryptographic key. Although most of the common and thoroughly examined algorithms in use today are public, this knowledge alone does not weaken security. Thus, key management vulnerabilities and associated threats are independent from the cryptographic countermeasures used. For Encryption, the Domain Model defines the Client, Encrypted Information, Sensitive Information, and Encryption Engine model elements.

One mechanism to provide data integrity and protect against non-repudiation is Digital Signatures. Non-repudiation is a system property that, in the case of a conflict or contention, can provide proof of a Subject and the systems actions at prior given time. Strong non-repudiation is a large advantage in both liability and enforcement standpoints. In addition, effective Digital Signature countermeasures must use non-forgeable signatures and provide verification for signature recipients. In the Domain Model, Digital Signatures guarantee that a particular client, who possesses a unique public/private key pair, examined and signed a particular electronic message. Additionally key signing ensures that no one has changed the message since its signing. For a Digital Signature system or component, first a client signs a message by generating a hash of the message, encrypting that hash with their secret (private) key, and appending the encrypted hash (the *signature*) to the message. On the receiving end, the clients also generates a similar hash of the message and decrypts the signature with the public key of the sender. If the hashes are equal, the message is valid. If they do not match, then the receiver knows the message was changed, or that the sender or the receiver has an erroneous key pair.

In the Domain Model, Encryption countermeasures secure confidentiality and a degree of integrity to system processes and resources. These countermeasures employ either Symmetric or Asymmetric encryption. In symmetric encryption, both transformations from cyphertext to plaintext and plaintext to cyphertext use the same exact key. Therefore, this type of encryption requires that only authorized clients possess or receive the shared secret key. Inherently, this type of system requires more trust than others do because once an unauthorized client possesses a key the security mechanisms are defeated. Asymmetric encryption uses two keys. The receiving client generates two related keys – one public and sent via open channels. The other – the secret key – is kept in a secure location. For this encryption type, the sending client encrypts a message using the public key of the receiver. The receiving client then uses their secret key, the other key in the pair, to decrypt the message into plaintext. Because of the algorithms involved, it is “very hard” to derive one key in the pair from the other pair key.[23]

Additionally, the Domain Model recognizes Mixed Encryption collaborations. This combines symmetric and asymmetric encryption yielding asymmetric encryption features but with the greater efficiency of symmetric encryption. In this hybrid, the sender encrypts the message using a symmetric key. Using asymmetric cryptography, the secret key of the sender encrypts and appends the asymmetrically encrypted symmetric key to the message. After receiving the message, the process reverses as the receiver decrypts with the public key of the sender to obtain the symmetric key. The symmetric key then decrypts the message.

Another type of countermeasure available to Encryption collaborations in the Domain Model is Public Key Infrastructure (PKI). These countermeasures use public key encryption for authentication as well as data and system integrity. For PKI mechanisms, Certification Authorities (CAs) perform similar tasks to key management but handle certificates as well. Certificates and PKI can permit clients to verify message integrity (with digital signatures) and use asymmetric encryption to keep messages confidential. PKI clients obtain certificates by requesting a Certificate Authority (CA) to certify a public key provided by the client. The CA verifies the client identity depending on policy procedures for generating the type of certificate requested. Many different standards and types of certificates are available, which provide differing levels of protection. After client verification, the CA generates a certificate that contains the public key of the client. The CA then digitally signs the certificate and sends it back to the client.

Intrusion Detection

In general, the term Intrusion Detection refers to system functions that provide monitoring, recording, and alarm functions that continuously check for misuse and attack. Intrusion Detection countermeasures are often manifested as network boundary devices, or as devices or hosts that passively monitor unauthorized access attempts and activities on a network or system. Some current Intrusion Detection Systems (IDS) can detect and attempt to prevent attacks in real time. Early warning and proactive solutions help prevent attacks from spreading further to other systems and networks.[24] Other IDS systems attempt to provide information and determine attack patterns after the fact. These types of collaborations help educate defenders and can lead to attack prevention in the future. A third IDS type is “Expert” based. Expert system technologies exploit data-mining techniques to detect and prevent attacks. IDS Signatures of “normal,” under attack and current status can allow software-assisted discovery of possible and potentially ongoing attacks. In addition to these types of IDS, this Domain Model collaboration

[23] “Very hard” is somewhat of an understatement. While the process is theoretically possible, it is “very hard” because the amount of computation required is not practical because it would take too long, typically centuries, and be so very expensive as to rival the economies of some nations.
[24] This is especially important in stopping flooding attacks whereby a machine is overwhelmed by too many connection requests, access attempts, etc., leaving it unable to do its intended work.

includes Audit trail countermeasures that might be present in various system security components, e.g., firewalls, operating systems, or network routers. In most cases Intrusion Detectors, either take an approach of signaling normal behavior deviations, or look specifically at previous known attack patterns. Some more sophisticated IDS use a hybrid of the two, either with uniform or separate component policy mechanisms to govern each. Policy countermeasures play a large issue in IDS collaborations because some policies may only detect or search for specific attacks or might be rendered ineffective if scaled back due to too many false positives.[25] Additionally, the Domain Model includes Intrusion Detection Policy, Intrusion Detection Engine, Alarm Condition, and Sensor model elements.

Concluding the Domain Model

Upon successful completion of both the countermeasures and their collaborations, the final step for the Domain Model was the development of a submodel defining Policy for the enumerated model elements. The Policy Model, modeled as a complex class diagram,[14] contains representations for the elements and characteristics of a security policy. In addition, the submodel project incorporates concepts of policy hierarchy, modification, traceability to requirements, and assurance of resulting policy. The submodel expresses security policy as a composite of written and its implementing policy. Modeling policy as this kind of relationship allows abstract implementing policies to provide a diverse array of security policies, optionally shaped by countermeasures and their respective attributes. The added development and inclusion of this Policy Model is important because it extends IA policy concepts into UML, further illuminating the complex issues of policy management and its associated problems to non-IA system designers.

*5.2. Applying the Profile – the Demo (Books On Line) and Target (Ultra*Log) Applications*

After elaborating a set of typical countermeasures (the Domain Model) and the specification for describing the security features of a system (the IA UML Profile,) the project turned to applying the profile to a demonstration application. Since, by this time, the project had joined with the Ultra*Log program, the choice was clear – use the Books On Line (BOL) demonstration application developed for the program earlier. The BOL application mimics an online book dealer, similar to Amazon.com. All program participants were already familiar with the application. The program's members had already seen the how the BOL application related to their ultimate application in the Ultra*Log framework.

[25] In the IDS context, false positives often occur from too finely tuned policies, resulting in threat and alarm conditions despite the absence of an actual attack. Also, when base-lining normal behavior, intrusion detectors risk acquiring an incorrect "normal" behavioral signature if it determines the signature while under undetected attack.

First, the project hired an outside consultant to reverse-engineer the BOL application. This produced an object model describing the BOL application's design. Next, the plan was to enhance the model with elements of the IA UML profile that illustrated how one could describe the BOL applications security countermeasures and attributes. Lastly, the project would apply the profile to the design of the Ultra*Log system. The Ultra*Log program would be undergoing enhancement of its largely missing security features. This would be the ideal time to see how IA-enhanced UML could aid the effort.

Unfortunately, the BOL application (like the Ultra*Log application of that time) had little in the way of security features. The project would have to enhance the BOL application with countermeasures and then analyze them for effectiveness. The project began concurrent exploratory efforts to add these enhancements to the BOL demo and to begin to work with the Ultra*Log security team, whose efforts had been underway for quite some time.

At about this point in time, circumstances overtook the IA UML project. The program team and its management saw that they would have to make changes in the program's overall direction, necessitating a change in funding priorities. Management agreed that the best course of action would be to bring an early end to the project's work in order to fund other work more central to the program's goals. Together, they planned an orderly end to the IA UML work. Consequently, further development, including the application to the Ultra*Log analysis and design, was halted. Efforts began to present the project's work-to-date in order to transfer its technology to those who might be able to continue the work – in industry and in subsequent research programs.

6. Technology Transfer

In the final phases of the project, attention turned to "getting out the word." Severely limited in resources, the project chose two opportunities that arose. The project's contributor from Rational Corporation, Jim Conallen, obtained a slot on the agenda of that company's annual conference, the Rational Users' Group Conference, to be held in July 2002 in Denver, CO. In September 2001, another team member, Brent Whitmore, requested and was granted the opportunity to present the project's work at the regularly scheduled Technical Meeting of the Object Management Group in Anaheim, CA. These meetings are the forum for OMG members where they determine changes to the standards that the group administers, include the UML standard.

6.1. Presentation: 2001 Rational Users' Group Conference

On July 25, 2001, Jan Filsinger and Jim Conallen presented the project's work to the Rational User's Group under the title "Modeling Information Assurance (IA.)" Due to a scheduling conflict with a very popular speaker, the session attendance was light. However, those attending expressed considerable interest in the work and showed general support for its approach.

6.2. Presentation: OMG Technical Meeting

In January 2002, Brent Whitmore presented a slightly revised version of Jan and Jim's August presentation to the OMG's Security Special Interest Group. The revisions mostly provided for comment from the audience and a solicitation for help in carrying the work forward.

The presentation elicited an hour and a half of follow-up questioning and discussion. The idea of modeling security within a software design seemed surprisingly new to this rather seasoned group of security professionals, many possessing decades of "hard" assurance experience from some of the most security-conscious parts of government and industry. Many were just unaware of how UML and modeling worked. Others had thought about it before, but had never had the time or the resources to devote to actually doing such a thing. Questions seemed evenly split between how UML and UML-employing methodologies worked, and how our particular methods and notational extensions fit into UML and UML-employing analysis and design methodologies.

Part of the presentation sought out specific comment on the work. The comments were quite favorable. The group has addressed some of the most difficult problems of securing distributed systems in the CORBA specification, seemed particularly taken by the ability to denote countermeasures in designs and to make reasonable arguments for their effectiveness. They commented that they would like to see a way for optional expression of threats and attacks and their relationships to the extension's existing IA features – countermeasures, assurance arguments, etc. They also suggested increased support for modeling boundaries and boundary controllers. They complained that "encryption" should really be called "data protection," a more general term. They also added that the profile should add ways to characterize the pedigree and quality-of-service of software and operating software facilities, such as a secure transports and trusted hosts.

Unfortunately, despite a plea for help, no one volunteered to shoulder the job of carrying the discussion and work into the future.

7. Future Work

The project successfully completed several milestones, but like security work in general, should be an ongoing process. Therefore, there are still several areas for future development.

The IA methodologies developed by the project team indicate several major areas for extension work. Discussed prior, current areas like Threat modeling show potential for further expansion and can offer added advantages to UML designers. Additionally, further elaboration of Threat → Vulnerability → Attack → Countermeasure models is a possibility as well. We should particularly incorporate the elements of the Countermeasure Characterization process that Network Associates Laboratories has so successfully applied in the past to other DARPA IA programs into IA UML methodology and notation.

In conjunction with this step, we should revise and publish a more concrete specification for IA UML methodology and then subject it to Peer Review for further refinement. We feel that examination of the methodology specification and consequent insight from external sources helps to build community familiarity, support, and desire for the overall IA extensions as well as strong technology.

We should expand the current Domain Model. Incorporating new or enhanced collaborations and model elements can make the Domain Model more robust and useful to both IA experts and software engineers. In addition, model integration with methodology improvements can help justify the results of these changes. Completing documentation of the Domain Model collaborations using UML diagrams incorporating Profile features is another venue for work.

The Profile is also a target for future progress. Opportunities exist to expand the Profile further, such as by defining specific Threat model elements or including common, but presently lacking, elements like non-repudiation.

One goal in the progression of the project was to apply the IA UML extensions to designs in order to help refine and verify the latest work as well as build confidence in the project mission. Although work began on applying the Profile to a design, further completion is required. Work towards successfully applying the IA extensions to a realistic project would be advantageous. The Ultra*Log project may still be a viable candidate. However, plans for potential work should not rule out other candidates and need not be constrained to the original ones attempted by the project team. We are pleased to note that the DARPA OASIS Demonstration and Validation Program, just begun, plans to extend and apply IA UML notation and methods to parts of their work.

Ideally, IA UML development would iterate several times over a few project applications with an end goal of becoming an OMG standard. Although the UML core is fixed, the OMG administration can and will adopt well-defined and useful extensions for UML. For this to happen however, the IA UML team should consciously follow and participate in all of the OMG adoption process stages.

8. Conclusions and Recommendations

Although the Domain Model and Profile offer several opportunities for improvement and expansion, the completed work shows real promise. The current IA extensions are a good start in providing “hard” assurance functionality to IA experts and system designers. Additionally, inclusion of team members from both the IA and software engineering communities was a large advantage to the project.

The Profile would benefit from incorporation of several IA concepts already present in the current Domain Model but not in the Profile. It would also benefit from some rewording of terms. A single notion of encryption should be replaced with separate notions of data protection and non-repudiation.

IA UML’s underlying methodology needs to be expressed more precisely and understandably. A single specification would benefit its users greatly. The methodology and notation should expand its support for modeling threats as well.

The team recommends future work towards further fulfilling IA UML goals. Specifically, the “Future Work” sections above outline some good starting points for additional work. Because a lot of development has gone into the creation of the Domain Model and Profile, future work should particularly emphasize the application of IA UML technology to actual system analysis and designs. To realize IA UML standardization, goals, the project team recommends several iterative applications of the IA UML Profile and methods, with refinement of both between application steps.

Finally, the project team encourages current developers to actively use the currently available IA UML extensions, or a similar extension, when they design systems with important security needs.